and the compiler phase in which the node was merged is marked. Finally, the action is marked as MERGED 1335.

Figure 14 is a flowchart illustrating annotation branch inversion according to one embodiment of the present invention. First, a new action node is created 1405. The major ID from the action node of the branch being inverted is copied to the new action node 1310. A new action number is assigned to the new node 1315. Next, the previous actions pointer of the new node is set to the action node of the branch being inverted and the compiler phase in which the node was merged is marked. Finally, the action is marked as BRANCH_INVERTED 1335.

CLAIMS

What is claimed is:

1. A method comprising:

installing a program onto a target machine;

executing the program; and

responsive to a change in profile data collected while the program executes which exceeds a predetermined threshold, recompiling the program while the target machine is idle.

2. The method of claim of claim 1, wherein said installing further comprises:

installing a continuous compiler;

sub A7

installing a runtime monitor;

copying an intermediate representation of a source file to the target machine;

building a profile database; and

compiling the intermediate representation to create an executable file.

3. The method of claim 1, wherein said executing further comprises:

running an executable version of the program;

collecting samples of process information at a controlled rate; and

while the target machine is idle, generating binary level and high level profiles.

4. The method of claim 1, wherein recompiling further comprises customizing compiler optimizations based on profile data generated during program execution.

5. The method of claim 4, wherein said customizing compiler optimizations is performed using annotations in a high level representation of an executable program which relate portions of the executable to the high level representation.

6. The method of claim 6, wherein creating said annotations comprises:

creating a new action node;

assigning to the new action node a major ID from a precomputed ID;

assigning a new action number to the new action node;

setting a previous action node pointer of the new action node to NULL;

marking a compiler phase in which the new node was created; and

marking an action of the new node as created.

7. The method of claim 6, wherein duplicating an annotation comprises:

creating two new action nodes;

Sub A6

copying a major ID to the new action nodes from an action node of instructions being copied;

assigning new action numbers to the two new nodes;

setting previous action node pointers of the new nodes to the action node being copied;

marking a compiler phase in which the nodes was duplicated; and

marking an action of the new nodes as duplicated.

8. The method of claim 6, wherein deleting an annotation comprises:

creating a new action node;

copying a major ID from an action node of an instruction being deleted to the new action node;

assigning a new action number to the new action node;

setting a previous action pointer in the new action node to the action node of the instruction being deleted;

marking a compiler phase in which the node was deleted; and

marking an action of the deleted node as deleted.

9. The method of claim 6, wherein merging of annotations comprises:

creating a new action node;

copying a major ID from a previous action node of instructions being merged;

assigning a new action number to the new action node;

setting a previous action pointer of the new node to a list of nodes of the instruction being merged;

Sub A7

adding the new action to a next actions pointer list of previous actions;

marking a compiler phase in which the node was merged;

marking an action of the new node as merged.

10. The method of claim 6, wherein annotation branch inversion comprises:

creating a new action node;

copying a major ID from a branch instruction being inverted;

assigning a new action number to the new node;

setting a previous action pointer of the new node to a node of a branch being inverted;

marking a compiler phase in which the inversion occurred; and

marking the branch as inverted.

11. A computer readable medium having stored thereon data representing sequences of instructions, the sequences of instructions which, when executed by a processor, cause the processor to perform the steps of:

installing a program onto a target machine;

executing the program; and

responsive to a change in profile data collected while the program executes which exceeds a predetermined threshold, recompiling the program while the target machine is idle.

12. The computer readable medium of claim of claim 11, wherein said installing further comprises:

installing a continuous compiler;

Sub A7

installing a runtime monitor;

copying an intermediate representation of a source file to the target machine;

building a profile database; and

compiling the intermediate representation to create an executable file.

13. The computer readable medium of claim 11, wherein said executing further comprises:

running an executable version of the program;

collecting samples of process information at a controlled rate; and

while the target machine is idle, generating binary level and high level profiles.

14. The computer readable medium of claim 11, wherein recompiling further comprises customizing compiler optimizations based on profile data generated during program execution.

15. The computer readable medium of claim 14, wherein said customizing compiler optimizations is performed using annotations in a high level representation of an executable program which relate portions of the executable to the high level representation.

16. The computer readable medium of claim 16, wherein creating said annotations comprises:

creating a new action node;

assigning to the new action node a major ID from a precomputed ID;

assigning a new action number to the new action node;

setting a previous action node pointer of the new action node to NULL;

Sub A7

marking a compiler phase in which the new node was created; and

marking an action of the new node as created.

17. The computer readable medium of claim 16, wherein duplicating an annotation comprises:

creating two new action nodes;

copying a major ID to the new action nodes from an action node of instructions being copied;

assigning new action numbers to the two new nodes;

setting previous action node pointers of the new nodes to the action node being copied;

marking a compiler phase in which the nodes was duplicated; and

marking an action of the new nodes as duplicated.

18. The computer readable medium of claim 16, wherein deleting an annotation comprises:

creating a new action node;

copying a major ID from an action node of an instruction being deleted to the new action node;

assigning a new action number to the new action node;

setting a previous action pointer in the new action node to the action node of the instruction being deleted;

marking a compiler phase in which the node was deleted; and

marking an action of the deleted node as deleted.

Sub A7

19. The computer readable medium of claim 16, wherein merging of annotations comprises:

creating a new action node;

copying a major ID from a previous action node of instructions being merged;

assigning a new action number to the new action node;

setting a previous action pointer of the new node to a list of nodes of the instruction being merged;

adding the new action to a next actions pointer list of previous actions;

marking a compiler phase in which the node was merged;

marking an action of the new node as merged.

20. The computer readable medium of claim 16, wherein annotation branch inversion comprises:

creating a new action node;

copying a major ID from a branch instruction being inverted;

assigning a new action number to the new node;

setting a previous action pointer of the new node to a node of a branch being inverted;

marking a compiler phase in which the inversion occurred; and

marking the branch as inverted.

21. A system comprising:

a storage device having stored therein a routine for transparent continuous compilation;

Sub A6 7

a processor coupled to the storage device which when executing the routine:

installs a program onto a target machine;

executes the program; and

responsive to a change in profile data collected while the program executes which exceeds a predetermined threshold, recompiles the program while the target machine is idle.

22. The system of claim of claim 21, wherein said installation further comprises:

installing a continuous compiler;

installing a runtime monitor;

copying an intermediate representation of a source file to the target machine;

building a profile database; and

compiling the intermediate representation to create an executable file.

23. The system of claim 21, wherein said execution further comprises:

running an executable version of the program;

collecting samples of process information at a controlled rate; and

while the target machine is idle, generating binary level and high level profiles.

24. The system of claim 21, wherein recompilation further comprises customizing compiler optimizations based on profile data generated during program execution.

25. The system of claim 24, wherein said customizing compiler optimizations is performed using annotations in a high level representation of an executable program which relate portions of the executable to the high level representation.

26. The system of claim 26, wherein creating said annotations comprises:

Sub A6

creating a new action node;

assigning to the new action node a major ID from a precomputed ID;

assigning a new action number to the new action node;

setting a previous action node pointer of the new action node to NULL;

marking a compiler phase in which the new node was created; and

marking an action of the new node as created.

27. The system of claim 26, wherein duplicating an annotation comprises:

creating two new action nodes;

copying a major ID to the new action nodes from an action node of instructions being copied;

assigning new action numbers to the two new nodes;

setting previous action node pointers of the new nodes to the action node being copied;

marking a compiler phase in which the nodes was duplicated; and

marking an action of the new nodes as duplicated.

28. The system of claim 26, wherein deleting an annotation comprises:

creating a new action node;

copying a major ID from an action node of an instruction being deleted to the new action node;

assigning a new action number to the new action node;

setting a previous action pointer in the new action node to the action node of the instruction being deleted;

Sub A7

marking a compiler phase in which the node was deleted; and

marking an action of the deleted node as deleted.

29. The system of claim 26, wherein merging of annotations comprises:

creating a new action node;

copying a major ID from a previous action node of instructions being merged;

assigning a new action number to the new action node;

setting a previous action pointer of the new node to a list of nodes of the instruction being merged;

adding the new action to a next actions pointer list of previous actions;

marking a compiler phase in which the node was merged;

marking an action of the new node as merged.

30. The system of claim 26, wherein annotation branch inversion comprises:

creating a new action node;

copying a major ID from a branch instruction being inverted;

assigning a new action number to the new node;

setting a previous action pointer of the new node to a node of a branch being inverted;

marking a compiler phase in which the inversion occurred; and

marking the branch as inverted.

Add A7